**Numbers**

isNumeric()

isNaN(str)

isNaN(n)

**isFinite(n)**

**parseInt (n)**

**parseFloat(n)**

number.toLocaleString()

### [num.toFixed(precision)](https://learn.javascript.ru/number#num-tofixed-precision)

**Функция isFinite(n) преобразует аргумент к числу и возвращает true, если это не NaN/Infinity/-Infinity:**

alert( isFinite(1) ); // true

alert( isFinite(Infinity) ); // false

alert( isFinite(NaN) ); // false

Значение NaN – единственное в своем роде, которое не равно ничему, включая себя.

В мире HTML/CSS многие значения не являются в точности числами. Например метрики CSS: 10pt или -12px.

Оператор '+' для таких значений возвратит NaN:

alert(+"12px") // NaN

Для удобного чтения таких значений существует функция [parseInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt):

alert( parseInt('12px') ); // 12

**Функция parseInt и ее аналог parseFloat преобразуют строку символ за символом, пока это возможно.**

Для проверки строки на число можно использовать функцию isNaN(str).

Она преобразует строку в число аналогично +, а затем вернёт true, если это NaN, то есть если преобразование не удалось

Однако, у такой проверки есть две особенности:

1. Пустая строка и строка из пробельных символов преобразуются к 0, поэтому считаются числами.
2. Если применить такую проверку не к строке, то могут быть сюрпризы, в частности isNaN посчитает числами значения false, true, null, так как они хотя и не числа, но преобразуются к ним.
3. function isNumeric(n) {
4. return !isNaN(parseFloat(n)) && isFinite(n);
5. }

**[Округление](https://learn.javascript.ru/number" \l "%D0%BE%D0%BA%D1%80%D1%83%D0%B3%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5)**

**Math.floor**

**Math.ceil**

**Math.round**

**Округление битовыми операторами**

alert( ~~12.3 ); // 12

Любая побитовая операция такого рода подойдет, например XOR (исключающее ИЛИ, "^") с нулем:

alert( 12.3 ^ 0 ); // 12

alert( 1.2 + 1.3 ^ 0 ); // 2, приоритет ^ меньше, чем +

### [Округление до заданной точности](https://learn.javascript.ru/number" \l "%D0%BE%D0%BA%D1%80%D1%83%D0%B3%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%BE-%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D0%BE%D0%B9-%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8)

Для округления до нужной цифры после запятой можно умножить и поделить на 10 с нужным количеством нулей. Например, округлим 3.456 до 2-го знака после запятой:

var n = 3.456;

alert( Math.round(n \* 100) / 100 ); // 3.456 -> 345.6 -> 346 -> 3.46

Таким образом можно округлять число и вверх и вниз.

### [num.toFixed(precision)](https://learn.javascript.ru/number" \l "num-tofixed-precision)

## [Неточные вычисления](https://learn.javascript.ru/number" \l "%D0%BD%D0%B5%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)

1. Сделать их целыми, сложить, а потом поделить:

alert( (0.1 \* 10 + 0.2 \* 10) / 10 ); // 0.3

Это работает, так как числа 0.1\*10 = 1 и 0.2\*10 = 2 могут быть точно представлены в двоичной системе.

1. Сложить, а затем округлить до разумного знака после запятой. Округления до 10-го знака обычно бывает достаточно, чтобы отсечь ошибку вычислений:
2. var result = 0.1 + 0.2;

alert( +result.toFixed(10) ); // 0.3